Incentive-Compatible Online Mechanisms for Resource Provisioning and Allocation in Clouds

Lena Mashayekhy, Mahyar Movahed Nejad, Daniel Grosu
Department of Computer Science
Wayne State University
Detroit, MI 48202, USA
{mlena, mahyar, dgrosu}@wayne.edu

Athanasios V. Vasilakos
Department of Computer Science
University of Western Macedonia
Greece
vasilako@ath.forthnet.gr

***Abstract*—Cloud providers provision their various resources such as CPUs, memory, and storage in the form of Virtual Machine (VM) instances which are then allocated to the users. We design online mechanisms for VM provisioning and allocation in clouds that consider several types of available resources. Our proposed online mechanisms make no assumptions about future demand of VMs, which is the case in real cloud settings. The proposed mechanisms are invoked as soon as a user places a request or some of the allocated resources are released and become available. The mechanisms allocate VM instances to selected users for the period they are requested for, and ensure that the users will continue using their VM instances for the entire requested period. In addition, the mechanisms determine the payment the users have to pay for using the allocated resources. We prove that the mechanisms are incentive-compatible, that is, they give incentives to the users to reveal their true valuations for their requested bundles of VM instances. We investigate the performance of our proposed mechanisms through extensive experiments.**

***Keywords*-cloud computing; online truthful mechanism; resource allocation.**

I. Introduction

Designing efficient mechanisms for Virtual Machine (VM) provisioning and allocation is a major problem in cloud computing. Such mechanisms should consider economic incentives for both cloud users and cloud providers in finding the market equilibrium.Current cloud providers such as Amazon EC2 and Microsoft Azure employ fixed-price and auction-based mechanisms in order to provision resources in the form of VM instances and sell them to the users. These mechanisms are offline mechanisms, thus they need to collect the information about all user requests and then decide the allocation of VM instances to users. However, cloud users request VM instances over time, thus, creating an online setting for the provisioning and allocation problem. Therefore, cloud providers need to design online provisioning and allocation mechanisms in order to provide faster services to the users and allocate their resources efficiently.

In this paper, we design mechanisms for the VM provisioning and allocation problem in clouds in the presence of multiple types of resources (e.g., cores, memory, storage, etc.). Our proposed mechanisms are online and thus, make no assumptions about future demand and supply of VMs, which is the case in real cloud settings. Online mechanisms calculate the allocation and payment as users arrive at the system and place their requests. Therefore, the cloud provider provisions and allocates VM instances as the resources become available.

In online settings, each user submits a bid for a bundle of VM instances, and specifies the amount of time the bundle must be allocated and a deadline. Each user has a private value (private *type*) for her requested bundle. The users are also selfish in the sense that they want to maximize their own utility. It may be beneficial for them to manipulate the system by declaring a false type (i.e., different bundles or bids from their actual request). One of the key properties of our proposed mechanisms is to give incentives to users so that they reveal their true valuations for their requested bundles. The objective of the mechanisms is to allocate resources to the users such that the social welfare (i.e., the sum of users' valuations for the requested bundles of VMs) is maximized. The mechanisms also determine the payments for each user.

Our Contribution. We address the problem of online VM provisioning and allocation in clouds in the presence of multiple types of resources. This is a strongly NP-hard problem. We design an offline incentive-compatible mechanism and a family of online incentive-compatible mechanisms for VM provisioning and allocation that give incentives to the users to reveal their true valuations for their requested bundles of VM instances. The proposed offline mechanism is optimal given that the information on all the future requests is known a priori. However, our proposed online mechanisms make no assumptions about future demand of VMs, which is the case in real cloud settings. Our proposed online mechanisms are invoked as soon as a user places a request or some allocated resources are released and become available. The mechanisms not only provision and allocate resources dynamically, but also determine the users' payments such that the incentive-compatibility property is guaranteed. The

proposed online mechanisms provide very fast solutions making them suitable for execution in real-time settings. We perform extensive experiments showing that the proposed online mechanisms are able to find near optimal allocations while satisfying the incentive-compatible property.

Related Work. Due to space limitation we will only discuss briefly the closest related work. The reader is referred to Parkes [1] for an introduction to online mechanisms. Online variants of Vickrey-Clarke-Groves (VCG) mechanisms were proposed by Gershkov and Moldovanu [2] and by Parkes and Singh [3]. These mechanisms focus on Bayesian-Nash incentive compatibility. However, these studies rely on a model of future availability, as well as future supply.

The problem of resource provisioning and allocation in clouds has been investigated by several researchers. Kuo et al. [4] proposed a 3-approximation algorithm for the VM placement problem to minimize the maximum access latency. Leslie et al. [5] proposed a framework for resource allocation and job scheduling of VMs aiming to cost efficiently execute deadline-constrained jobs. In our previous studies, we proposed truthful mechanisms for VM allocation in clouds in periodic-time settings [6], [7]. However, none of these studies consider online settings. Zhang et al. [8] proposed an online auction mechanism for resource allocation in clouds. The preemption is not allowed, and there are assumptions that job lengths and bids are within known intervals. In addition, they considered only one type of resources. Zaman and Grosu [9] proposed a truthful online mechanism for provisioning and allocation of VM instances in clouds. However, their mechanism assumes that the cloud provider offers only one type of resources, computational resources. The current work is different from the two above-mentioned studies since it considers the existence of several resource types, being more suitable for use in real cloud settings. Note that considering one resource makes the problem NP-hard, while in our study, we tackle a much more challenging problem which is strongly NP-hard. In addition, satisfying incentive-compatibility in our settings brings about more challenges.

Organization. The rest of the paper is organized as follows. In Section II, we describe the online VM provisioning and allocation problem in clouds. In Section III, we introduce the basic concepts of mechanism design, and we present our proposed offline optimal mechanism. In Section IV, we present the proposed online mechanisms, and we characterize their properties. In Section V, we evaluate the mechanisms by extensive experiments. In Section VI, we summarize our results and present possible directions for future research.

II. VM Provisioning and Allocation Problem

We define the problem of online VM provisioning and allocation in clouds (OVMPAC) in the presence of multiple types of resources as follows. We consider a cloud provider offering a set of R types of resources, $\mathcal{R} = \{1, \ldots, R\}$, to users in the form of VM instances. These types of resources include cores, memory, storage, etc. The cloud provider has restricted capacity, C_r, on each resource $r \in \mathcal{R}$ available for allocation. The cloud provider offers these resources in the form of M types of VMs, $\mathcal{VM} = \{1, \ldots, M\}$, where each VM of type $m \in \mathcal{VM}$ provides a specific amount of each type of resource $r \in \mathcal{R}$. The amount of resources of type r that one VM instance of type m provides is denoted by w_{mr}. As an example, in Table I, we present the four types of VM instances offered by Amazon EC2 at the time of writing this paper. If we consider that CPU represents the type 1 resource, memory, the type 2 resource, and storage, the type 3 resource, we can characterize, for example, the Medium instance ($m = 2$) by: $w_{11} = 2$, $w_{12} = 3.75$ GB, and $w_{13} = 410$ GB.

Table I: VM instance types offered by Amazon EC2.

	Small $m = 1$	Medium $m = 2$	Large $m = 3$	Extralarge $m = 4$
CPU	1	2	4	8
Memory (GB)	1.7	3.75	7.5	15
Storage (GB)	160	410	850	1690

A set $\mathcal{U}$ of N users are requesting a set of VM instances for a certain amount of time in order to execute their applications (jobs) on the cloud. User i, $i \in \mathcal{U}$, requests a bundle $S_i = \langle k_{i1}, k_{i2}, \ldots, k_{iM} \rangle$ of M types of VM instances, where k_{im} is the number of requested VM instances of type $m \in \mathcal{VM}$. In addition, she specifies a bid b_i for her requested bundle S_i. User i is characterized by her type $\theta_i = (S_i, a_i, l_i, d_i, b_i)$, where a_i is the arrival time of her request, l_i is the amount of time for which the requested bundle must be allocated, and d_i is the deadline for her job completion. For example, type $(\langle 4, 3, 1, 2 \rangle, 2, 1, 7, \$15)$ represents a user requesting 4 Small VM instances, 3 Medium VM instances, 1 Large VM instance, and 2 Extra large VM instances; the request arrives at time 2, needs 1 unit of time to execute, expires at time 7, and her bid is \$15. We denote by $\sigma_{ir} = \sum_{m \in \mathcal{VM}} k_{im} w_{mr}$, the total amount of each resource of type r that user i has requested.

We define $\delta_i = d_i - l_i$ as the time by which S_i must be allocated to user i in order for her job to complete its execution. If the cloud provider allocates a requested bundle, the request is never preempted. User i values her requested bundle S_i at b_i, which is the maximum price a user is willing to pay for using the requested bundle if it is allocated within time window $[a_i, \delta_i]$. The users are assumed to be *single-minded*. That means, user i desires only S_i and derives a value of b_i if she gets S_i, or any superset of it, for the specified time before its deadline, and zero value, otherwise.

To design incentive-compatible mechanisms, we consider the standard mechanism design objective, that is, maximizing the social welfare [10]. Maximizing social welfare can help a cloud provider increase its revenue by allocating the VMs to the users who value them the most. We denote

by V the *social welfare*, which is defined as the sum of users' valuations, $V = \sum_{i \in \mathcal{U}} b_i \cdot x_i$, where x_i, $i \in \mathcal{U}$, are decision variables defined as follows: $x_i = 1$, if bundle S_i is allocated to user i within time window $[a_i, \delta_i]$; and $x_i = 0$, otherwise. Our goal is to design online incentive-compatible mechanisms maximizing V, that is, mechanisms that solve OVMPAC.

We also define the offline version of OVMPAC, called VMPAC, which considers that the information on all the future requests is known a priori. In order to formulate VMPAC as an integer program we define the decision variables over time $t \in \mathcal{T}$ as follows: $X_{it} = 1$, if S_i is allocated to i at t; and $X_{it} = 1$, otherwise. In addition, we define indicator parameters as follows: $y_{it} = 1$, if $a_i \leq t \leq \delta_i$; and $y_{it} = 1$, otherwise. The feasibility of the allocation to user i is indicated by y_{it}. This indicator parameter ensures that the allocation of the requested bundle is within time window $[a_i, \delta_i]$.

We formulate the problem of offline VM provisioning and allocation in clouds (VMPAC) as an Integer Program (called VMPAC-IP) as follows:

$$\text{Maximize} \sum_{i \in \mathcal{U}} \sum_{t \in \mathcal{T}} b_i \cdot y_{it} \cdot X_{it} \tag{1}$$

Subject to:

$$\sum_{t \in \mathcal{T}} X_{it} \leq 1, \forall i \in \mathcal{U} \tag{2}$$

$$\sum_{i \in \mathcal{U}} \sum_{\omega = t - l_i + 1}^{t} \sum_{m \in \mathcal{VM}} k_{im} w_{mr} y_{i\omega} X_{i\omega} \leq C_r, \quad \forall r \in \mathcal{R}, \forall t \in \mathcal{T} \tag{3}$$

$$X_{it} = \{0, 1\}, \forall i \in \mathcal{U}, \forall t \in \mathcal{T} \tag{4}$$

$$y_{it} = \{0, 1\}, \forall i \in \mathcal{U}, \forall t \in \mathcal{T} \tag{5}$$

The objective function is to maximize social welfare V, where $x_i = \sum_{t \in \mathcal{T}} y_{it} \cdot X_{it}$. Constraints (2) ensure that the request of each user is fulfilled at most once. Constraints (3) guarantee that the allocation of each resource type does not exceed the available capacity of that resource for any given time. Constraints (4) and (5) represent the integrality requirements for the decision variables and indicator parameters. These constraints force the cloud provider to provision the whole bundle of VM instances and to allocate bundles to the selected users. The VMPAC problem is strongly NP-hard by a simple reduction from the multidimensional knapsack problem [11]. Note that VMPAC-IP assumes that the information about all users' requests is available at the time of solving it. As a result, if solved, VMPAC-IP finds the optimal allocation of cloud resources in an offline setting. However, in an online setting, we do not have the information about future requests (such as arrivals), and thus, we have to rely on online mechanisms that solve the OVMPAC problem. Our goal is to design such online incentive-compatible mechanisms that solve the OVMPAC problem.

III. Mechanism Design Framework

In this section, we first present the basic concepts of mechanism design and then propose an offline optimal mechanism.

A. Preliminaries of Mechanism Design

In general, a deterministic mechanism $\mathcal{M}$, is defined as a tuple $(\mathcal{A}, \mathcal{P})$, where $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_N)$ is the allocation function that determines which users receive their requested bundles, and $\mathcal{P} = (\mathcal{P}_1, \ldots, \mathcal{P}_N)$ is the payment rule that determines the amount that each user must pay for the allocated bundles. In our model, each user $i \in \mathcal{U}$ is characterized by her true type denoted by θ_i. Each user's type is private knowledge. The users may declare different types from their true types. We denote by $\hat{\theta}_i = (\hat{S}_i, \hat{a}_i, \hat{l}_i, \hat{d}_i, \hat{b}_i)$ user i's declared type. Note that $\theta_i = (S_i, a_i, l_i, d_i, b_i)$ is user i's true type. The valuation function $v_i(\hat{\theta}_i)$ of user i is defined as follows: $v_i(\hat{\theta}_i) = b_i$, if S_i is allocated by $\mathcal{A}$ $\wedge (S_i \subseteq \hat{S}_i) \wedge (t_i \leq \delta_i)$; and $v_i(\hat{\theta}_i) = 0$, otherwise, where t_i is the time at which $\hat{S}_i$ has been allocated to user i. The goal is to design incentive-compatible mechanisms that maximize the social welfare V, where $V = \sum_{i \in \mathcal{U}} v_i(\hat{\theta}_i) \cdot x_i$.

The utility function of user i is *quasi-linear*, and thus, it is defined as the difference between her valuation and payment, $u_i(\hat{\theta}_i) = v_i(\hat{\theta}_i) - \mathcal{P}_i(\hat{\theta}_i)$, where $\mathcal{P}_i(\hat{\theta}_i)$ is the payment for user i calculated by the mechanism using the payment rule $\mathcal{P}$. The goal of a user is to maximize her utility, and she may manipulate the mechanism by lying about her true type to increase her utility. In our case, the type of a user consists of a bundle, an arrival time, an amount of time for which the requested bundle must be allocated, a deadline, and a value. As a result, a user can lie about any of these parameters in the hope to increase her utility. These manipulations by the users will lead to inefficient allocation of resources and ultimately will reduce the revenue obtained by the cloud provider. We want to prevent such manipulations by designing incentive-compatible mechanisms for solving OVMPAC. We denote by $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_N)$ the vector of types of all users. In addition, $\boldsymbol{\theta}_{-i}$ is the vector of all types except user i's type (i.e., $\boldsymbol{\theta}_{-i} = (\theta_1, \ldots, \theta_{i-1}, \theta_{i+1}, \ldots, \theta_N)$). A mechanism is *incentive-compatible* if all users have incentives to reveal their true types.

Definition 1 (Incentive compatibility): A mechanism $\mathcal{M}$ is *incentive-compatible* (or truthful) if for every user i, for every type declaration of the other users $\hat{\boldsymbol{\theta}}_{-i}$, a true type declaration θ_i and any other declaration $\hat{\theta}_i$ of user i, we have that $u_i(\theta_i, \hat{\boldsymbol{\theta}}_{-i}) \geq u_i(\hat{\theta}_i, \hat{\boldsymbol{\theta}}_{-i})$.

In other words, a mechanism is incentive-compatible if truthful reporting is a dominant strategy for the users, that is, the users maximize their utilities by truthful reporting

independently of what the other users are reporting. To obtain an incentive-compatible mechanism the allocation function $\mathcal{A}$ must be monotone and the payment rule must be based on the critical value.

For our model, we define monotonicity in terms of the following preference relation $\succeq$ on the set of types: $\hat{\theta}'_i \succeq \hat{\theta}_i$ if $\hat{S}_i \succeq \hat{S}'_i$, $\hat{a}'_i \leq \hat{a}_i$, $\hat{l}'_i \leq \hat{l}_i$, $\hat{d}'_i \geq \hat{d}_i$, and $\hat{b}'_i \geq \hat{b}_i$ for user i. Moreover, $\hat{S}'_i \succeq \hat{S}_i$ if $\sigma'_{ir} \leq \sigma_{ir}$, $\forall r \in \mathcal{R}$. That means the type $\hat{\theta}'_i$ is more preferred than $\hat{\theta}_i$ if user i requests a smaller bundle, submits an earlier request, the bundle for a shorter time period, a later deadline, and submits a higher value. In our setting, users cannot report an earlier arrival (i.e., $\hat{a}_i \leq a_i$), a shorter length (i.e., $\hat{l}_i \leq l_i$), or a later deadline (i.e., $\hat{d}_i \geq d_i$) than their true arrival time, true length, and true deadline. There is no reason for a user to submit her request earlier than when her job is ready for execution. Declaring a shorter length does not allow the completion of the job. Reporting a later deadline may result in getting her bundle too late to complete her job on time.

Definition 2 (Monotonicity): An allocation function $\mathcal{A}$ is *monotone* if it allocates the resources to user i with $\hat{\theta}_i$ as her declared type, then it also allocates the resources to user i with $\hat{\theta}'_i$, where $\hat{\theta}'_i \succeq \hat{\theta}_i$.

In other words, $\mathcal{A}$ is monotone if any winning user who receives her requested bundle by declaring a type $\hat{\theta}_i$ is still wining if she requests a more preferred type. Any incentive-compatible mechanism $\mathcal{M}$ has a payment rule $\mathcal{P}$ such that the payment of any user i, $\mathcal{P}_i$, is independent of her request.

Definition 3 (Critical value): Let $\mathcal{A}$ be a monotone allocation function, then for every θ_i, there exist a unique value b^c_i, called *critical value*, such that $\forall \hat{\theta}_i \succeq (S_i, a_i, l_i, d_i, b^c_i)$, $\hat{\theta}_i$ is a winning declaration, and $\forall \hat{\theta}_i \prec (S_i, a_i, l_i, d_i, b^c_i)$, $\hat{\theta}_i$ is a losing declaration.

The mechanism $\mathcal{M}$ works as follows. It first receives the declared types from each participating user, and then, based on the received types determines the allocation using the allocation function $\mathcal{A}$ and the payments using the payment rule $\mathcal{P}$. The payment rule $\mathcal{P}$ is based on the critical value and is defined as follows: $\mathcal{P}_i(\hat{\boldsymbol{\theta}}) = b^c_i$, if user i is allocated; and 0, otherwise. Here, b^c_i is the critical value of user i.

In the next subsection, we incorporate our proposed VMPAC-IP in the design of a VCG-based optimal mechanism which computes the allocation and payment offline.

B. Incentive-Compatible Offline Optimal Mechanism

We introduce a VCG-based incentive-compatible optimal mechanism that solves VMPAC, the offline version of OVMPAC problem. Since the setting is offline, the VCG-based mechanism has all the information about the users, and thus, it finds the optimal solution. A VCG-based mechanism [10] requires an optimal allocation algorithm implementing the allocation function $\mathcal{A}$ and a payment rule given by: $\mathcal{P}_i(\hat{\theta}_i) = \sum_{j \in \mathcal{A}(\hat{\boldsymbol{\theta}}_{-i})} v_j(\hat{\theta}_j) - \sum_{j \in \mathcal{A}(\hat{\boldsymbol{\theta}}), j \neq i} v_j(\hat{\theta}_j)$, where $\sum_{j \in \mathcal{A}(\hat{\boldsymbol{\theta}}_{-i})} v_j(\hat{\theta}_j)$ is the optimal social welfare that would have been obtained had user i not participated, and $\sum_{j \in \mathcal{A}(\hat{\boldsymbol{\theta}}), j \neq i} v_j(\hat{\theta}_j)$ is the sum of all users valuations except user i's.

Algorithm 1 VCG-VMPAC Mechanism ($\mathbf{C}$)

1: {Collect user requests offline (types).}
2: **for all** $i \in \mathcal{U}$ **do**
3: Collect user type $\hat{\theta}_i = (\hat{S}_i, \hat{a}_i, \hat{l}_i, \hat{d}_i, \hat{b}_i)$ from user i
4: {Allocation.}
5: (V^*, $\mathbf{x}^*$) = Solve IP-VMPAC($\hat{\boldsymbol{\theta}}, \mathbf{C}$)
6: Provisions and allocates VM instances according to $\mathbf{x}^*$.
7: {Payment.}
8: **for all** $i \in \mathcal{U}$ **do**
9: (V'^*, $\mathbf{x}'^*$) = Solve IP-VMPAC($\hat{\boldsymbol{\theta}}_{-i}, \mathbf{C}$)
10: $sum_1 = sum_2 = 0$
11: **for all** $j \in \mathcal{U}, j \neq i$ **do**
12: $sum_1 = sum_1 + \hat{b}_j x'^*_j$
13: $sum_2 = sum_2 + \hat{b}_j x^*_j$
14: $\mathcal{P}_i = sum_1 - sum_2$
15: **Output:** V^*; $\mathbf{x}^*$; $\mathcal{P} = (\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_N)$

We define the VCG-based mechanism that solves the VMPAC problem as follows:

Definition 4: The VCG-VMPAC mechanism consists of the optimal allocation algorithm that solves IP-VMPAC and the payment function defined by the VCG payment rule.

The VCG-VMPAC mechanism is given in Algorithm 1. VCG-VMPAC has one input parameter, the vector of resource capacities $\mathbf{C} = (C_1, \ldots, C_R)$, and three output parameters: V^*, the optimal social welfare, $\mathbf{x}^* = (x^*_1, x^*_2, \ldots, x^*_N)$, the optimal allocation of VM instances to the users, and $\mathcal{P}$ the payments. The mechanism collects the requests from the users, expressed as types (lines 1-3), and determines the optimal allocation by solving the IP-VMPAC given in Equations (1) to (5) (line 5). Once the optimal allocation is determined the mechanism provisions the required number and types of VM instances and determines the payments. The users are then charged the amount determined by the mechanism (lines 8-14). The VCG payment of a user i is calculated by solving the IP-VMPAC to find the allocation and welfare obtained without user i's participation (line 9). Based on the optimal allocation to the users with and without user i's participation, the mechanism finds the payment for user i, where sum_1 is the sum of all values without user i's participation in the mechanism, and sum_2 is the sum of all except user i's value in the optimal case (lines 10-14).

Being a VCG-based mechanism, VCG-VMPAC is incentive-compatible [10], and it determines the optimal allocation. However, the VMPAC is strongly NP-hard, and thus, the execution time of VCG-VMPAC becomes prohibitive for large instances of VMPAC. In addition, VCG-VMPAC computes the allocation and payment offline since it has all the information about future demands. However, in a real settings this information is not available to the cloud providers and requires designing online mechanisms.

IV. Online Mechanisms for VM Provisioning and Allocation

Our goal is to design incentive-compatible greedy mechanisms that solve the OVMPAC problem in online settings. The VM instances have R dimensions, where the R dimensions correspond to the R types of resources. Since the cloud provider provisions resources in the form of VM instances, any bundle of VMs can be seen as one R-dimensional item. Without loss of generality, we consider that the smallest item in the R-dimensional space contains one unit of each resources. This assumption does not restrict our proposed model since the resource capacities can be normalized to their units. As a result, the total volume of available items to allocate to the users is $\prod_{r\in\mathcal{R}} C_r$. In this section, we present a family of incentive-compatible online mechanisms for the OVMPAC problem, called OVMPAC-X.

A. OVMPAC-X Mechanisms

The OVMPAC-X family is given in Algorithm 2. The OVMPAC-X is an event handler, that is, it is invoked when a new user request arrives or some allocated VM instances become available. OVMPAC-X takes as input an Event, the current allocation set $\mathcal{A}$, and the payment set $\mathcal{P}$. An Event is either a release of resources or an arrival of a user request. In lines 1 to 8, OVMPAC-X sets the current time to t and initializes four variables as follows:

$\mathcal{Q}^t$: the set of types of the users that have not been allocated. Formally,
$\mathcal{Q}^t \leftarrow \{\hat{\theta}_i | i \in \mathcal{U}, t \leq \hat{\delta}_i \wedge \nexists t_i < t : (\hat{\theta}_i, t_i) \in \mathcal{A}\}$;
$\tilde{\mathcal{Q}}^t$: the set of types of the users that have been allocated and their jobs have not finished yet. Formally,
$\tilde{\mathcal{Q}}^t \leftarrow \{\hat{\theta}_i | i \in \mathcal{U} \wedge \exists t_i < t : (\hat{\theta}_i, t_i) \in \mathcal{A} \wedge t_i + \hat{l}_i > t\}$;
σ_{ir}: the amount of each resource of type r requested by user i; and,
C_r^t: the available capacity of the resource r at time t.

The mechanism stores the resource capacities at time t as a vector $\mathcal{C}^t$ (line 9). Then, it proceeds only if resources and requests are available. OVMPAC-X determines the allocation by calling OVMPAC-X-ALLOC. The allocation function OVMPAC-X-ALLOC returns $\mathcal{A}^t$, the set of users who would receive their requested bundles at time t (line 12). The mechanism then updates the overall allocation set $\mathcal{A}$ using the newly determined set $\mathcal{A}^t$. Then, the mechanism determines the payment of users in $\mathcal{A}^t$ by calling OVMPAC-X-PAY. The payment function OVMPAC-X-PAY returns set $\mathcal{P}^t$ containing the payment of users at time t (line 14). The mechanism updates the overall payment set $\mathcal{P}$ using the newly determined set $\mathcal{P}^t$ (line 15).

B. Allocation algorithms of OVMPAC-X

The allocation algorithm OVMPAC-X-ALLOC is given in Algorithm 3. We consider two allocation algorithms, OVMPAC-I-ALLOC, and OVMPAC-II-ALLOC, where the settings of the two algorithms differ from each other based on the length of jobs, and the type of time (discrete or continuous). Based on the settings of the two allocation algorithms, we define a metric called the *bid density*. OVMPAC-X-ALLOC algorithm allocates the VM instances to users in decreasing order of their bid densities. We define OVMPAC-I-ALLOC, and OVMPAC-II-ALLOC, as follows:

Algorithm 2 OVMPAC-X Mechanisms (Event, $\mathcal{A}, \mathcal{P}$)

1: $t \leftarrow$ Current time
2: $\mathcal{Q}^t \leftarrow \{\hat{\theta}_i | i \in \mathcal{U}$, i has not been allocated$\}$
3: $\tilde{\mathcal{Q}}^t \leftarrow \{\hat{\theta}_i | i \in \mathcal{U}$, ($i$ has been allocated) $\wedge$ (its job has not finished yet)$\}$
4: **for all** $i \in \mathcal{U}$ **do**
5: **for all** $r \in \mathcal{R}$ **do**
6: $\sigma_{ir} = \sum_{m\in\mathcal{VM}} k_{im} w_{mr}$
7: **for all** $r \in \mathcal{R}$ **do**
8: $C_r^t \leftarrow C_r - \sum_{i|\hat{\theta}_i \in \tilde{\mathcal{Q}}^t} \sigma_{ir}$
9: $\mathcal{C}^t \leftarrow (C_1^t, \ldots, C_R^t)$; vector of resource capacities at time t
10: **if** $\mathcal{Q}^t = \emptyset$ or $\mathcal{C}^t = \mathbf{0}$ **then**
11: **return**
12: $\mathcal{A}^t \leftarrow$ OVMPAC-X-ALLOC$(t, \mathcal{Q}^t, \mathcal{C}^t)$
13: $\mathcal{A} \leftarrow \mathcal{A} \cup \mathcal{A}^t$
14: $\mathcal{P}^t \leftarrow$ OVMPAC-X-PAY$(t, \mathcal{Q}^t, \mathcal{A}^t, \mathcal{C}^t)$
15: $\mathcal{P} \leftarrow \mathcal{P} \cup \mathcal{P}^t$
16: **return** $\mathcal{A}, \mathcal{P}$

1) OVMPAC-I-ALLOC: This algorithm considers the setting in which a set $\mathcal{U}$ of N users are requesting a heterogeneous set of VM instances for *one unit* of time in order to execute their applications/jobs on the cloud. It also considers a discrete-time model such that $t \in \{0, 1, \cdots, T\}$. In this case $l_i = 1$, and the bid density is:

$$f_i = \frac{\hat{b}_i}{\prod_{r\in\mathcal{R}} \sigma_{ir}} \tag{6}$$

2) OVMPAC-II-ALLOC: This algorithm considers the setting in which a set $\mathcal{U}$ of N users are requesting a heterogeneous set of VM instances for *any length* of time in order to execute their applications/jobs on the cloud. It also considers a continuous-time model such that $t \in [0, T]$. Note that the request time length for any user i is $\hat{l}_i \geq 1$. The bid density is defined as follows:

$$f_i = \frac{\hat{b}_i}{\hat{l}_i \cdot \prod_{r\in\mathcal{R}} \sigma_{ir}} \tag{7}$$

The bid of user i for a bundle of VM instances for time $\hat{l}_i$ can be interpreted as requesting a hyper-rectangle with volume $\hat{l}_i \cdot \prod_{r\in\mathcal{R}} \sigma_{ir}$ in the $(R+1)$-dimensional space defined by the R resource types and the time. User i values this volume at $\hat{b}_i$, if allocated. Hence, f_i represents how much user i values one unit of volume from the $(R+1)$-dimensional space. In this setting, we consider that the bids are chosen from an interval $[\underline{b}, \overline{b}]$ without assuming any distribution, where $\underline{b}$ and $\overline{b}$ are the minimum and maximum bids, respectively.

OVMPAC-X-ALLOC sorts all types in non-increasing order of bid densities, f_i (line 4). Then the algorithm

Algorithm 3 OVMPAC-X-ALLOC$(t, \mathcal{Q}^t, \mathcal{C}^t)$

1: $\mathcal{A}^t \leftarrow \emptyset$
2: **for all** $i|\hat{\theta}_i \in \mathcal{Q}^t$ **do**
3: $f_i = \frac{\hat{b}_i}{\prod_{r\in\mathcal{R}} \sigma_{ir}}$, for OVMPAC-I-ALLOC; or
$f_i = \frac{\hat{b}_i}{\hat{l}_i \cdot \prod_{r\in\mathcal{R}} \sigma_{ir}}$, for OVMPAC-II-ALLOC
4: Sort all $\hat{\theta}_i \in \mathcal{Q}^t$ in non-increasing order of f_i
5: **for all** $\hat{\theta}_i \in \mathcal{Q}^t$ in non-increasing order of f_i **do**
6: $\hat{\mathcal{C}} = \mathcal{C}^t$
7: $flag \leftarrow$ TRUE
8: **for all** $r \in \mathcal{R}$ **do**
9: $\hat{C}_r = \hat{C}_r - \sigma_{ir}$
10: **if** $\hat{C}_r < 0$ **then**
11: $flag \leftarrow$ FALSE
12: **break**;
13: **if** $flag$ **then**
14: $\mathcal{C}^t = \hat{\mathcal{C}}$
15: $\mathcal{A}^t \leftarrow \mathcal{A}^t \cup (\hat{\theta}_i, t)$
16: **Output:** $\mathcal{A}^t$

allocates bundles requested by the sorted users in $\mathcal{Q}^t$ while resources last (lines 5-15). The mechanism uses this ordering for allocation since the cloud provider is interested in users who want to pay more per unit of their resources per unit of time. OVMPAC-X-ALLOC tries to maximize the sum of the reported values of the users who get their requested bundles. Finally, OVMPAC-X-ALLOC returns the set $\mathcal{A}^t$ of users who are selected for allocation at time t. The time complexity of OVMPAC-X-ALLOC is $O(N(\log N + MR))$. This is because sorting the types requires $O(N \log N)$, while checking the feasibility of the allocation for each user requires $O(NMR)$.

C. Payment functions of OVMPAC-X

The payment function OVMPAC-X-PAY is given in Algorithm 4. This function calculates the *critical payment* of each user i if her requested bundle is allocated at t. The critical payment of user i is the minimum value that she must report to get her requested bundle at time t. OVMPAC-X-PAY determines the set $\hat{\mathcal{Q}}$ of types of users who are allocated or not allocated at t (line 1). This set does not include types of users who are allocated before t, and have not finished their jobs (i.e., their deadlines are not passed yet). OVMPAC-X-PAY calculates f_i for all users in $\hat{\mathcal{Q}}$ (lines 2-3). Then, OVMPAC-X-PAY determines the payment for all users that have been allocated at time t. In doing so, it updates the vector of capacities of resources $\hat{\mathcal{C}}$ to the capacities before allocating to user i (lines 5-7). Then, it calls the allocation algorithm, OVMPAC-X-ALLOC, without considering the participation of user i (line 9). Then, OVMPAC-X-PAY tries to find a user j who had not been allocated at t when user i participated, and would have been allocated at t if user i did not participate (lines 10-16). If OVMPAC-X-PAY finds such a user, it stores her index q (line 11), and it determines the payment of user i based on the density of user q (line 14); otherwise user i pays 0 (line 16). In other

Algorithm 4 OVMPAC-X-PAY$(t, \mathcal{Q}^t, \mathcal{A}^t, \mathcal{C}^t)$

1: $\hat{\mathcal{Q}} = \mathcal{Q}^t \cup \{\hat{\theta}_i | (\hat{\theta}_i, t) \in \mathcal{A}^t\}$
2: **for all** $i|\hat{\theta}_i \in \hat{\mathcal{Q}}$ **do**
3: $f_i = \frac{\hat{b}_i}{\prod_{r\in\mathcal{R}} \sigma_{ir}}$, for OVMPAC-I-ALLOC; or
$f_i = \frac{\hat{b}_i}{\hat{l}_i \cdot \prod_{r\in\mathcal{R}} \sigma_{ir}}$, for OVMPAC-II-ALLOC
4: **for all** $(\hat{\theta}_i, t) \in \mathcal{A}^t$ in non-increasing order of f_i **do**
5: $\hat{\mathcal{C}} \leftarrow \mathcal{C}^t$
6: **for all** $r \in \mathcal{R}$ **do**
7: $\hat{C}_r = \hat{C}_r + \sigma_{ir}$
8: $q = -1$;
9: $\bar{\mathcal{A}} \leftarrow$ OVMPAC-X-ALLOC$(t, \hat{\mathcal{Q}} \setminus \hat{\theta}_i, \hat{\mathcal{C}})$
10: **for all** $\hat{\theta}_j \in \mathcal{Q}^t \cap \{\hat{\theta}_j | (\hat{\theta}_j, t) \in \bar{\mathcal{A}}\}$ in non-increasing order of f_j, where $f_j < f_i$ **do**
11: $q = j$;
12: **break**;
13: **if** q **then**
14: $\mathcal{P}_i^t \leftarrow f_q \cdot \hat{l}_i \cdot \prod_{r\in\mathcal{R}} \sigma_{ir}$
15: **else**
16: $\mathcal{P}_i^t \leftarrow 0$
17: **Output:** $\mathcal{P}^t = (\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_N)$

words, the payment of user i is calculated by multiplying $\hat{l}_i \cdot \prod_{r\in\mathcal{R}} \sigma_{ir}$ with the highest density among losing users, (i.e., that of user q), who would win if user i would not participate. This is the minimum value to be reported by user i such that she gets her requested bundle. Finally, the set $\mathcal{P}^t$ is returned to the mechanism.

D. Incentive-compatibility of OVMPAC-X

In order to prove that the mechanisms are incentive-compatible, we need to show that the allocation algorithms are monotone, and the payment functions are based on the critical value.

Theorem 1: OVMPAC-X mechanisms are incentive-compatible.

Proof: (Sketch) We first show that the allocation algorithm OVMPAC-X-ALLOC is monotone. If user i wins by reporting $\hat{\theta}_i$, then she will also win if she reports a more preferred type $\hat{\theta}'_i \geq \hat{\theta}_i$. Clearly, if user i reports $\hat{b}'_i \geq \hat{b}_i$, her bid $\hat{\theta}'_i$ will be allocated if $\hat{\theta}_i$ is also allocated. Similarly, if a user gets the allocation by reporting $\hat{d}_i$, she will also get it by reporting $\hat{d}'_i \geq \hat{d}_i$. Similar reasoning applies for the other parameters in the type of the user.

We now show that the payment function implemented by OVMPAC-X-PAY is based on the critical value. The payment function computes the minimum value that the users must report to get the allocation. As a result, if user i reports a bid below the minimum value, she loses; otherwise she wins. This unique value is the critical value for user i.

Since the payment is the critical value payment and the allocation function is monotone, it follows from Parkes [1] that OVMPAC-X are incentive-compatible. ■

V. Experimental Results

We perform extensive experiments with real workload data in order to investigate the properties of our proposed

Table II: Statistics of workload logs.

Logfile	Avg jobs/ hour	Range of CPU	Range of memory (MB)	Range of Storage (MB)	Available CPUs	Memory Capacity (MB)	Storage Capacity (MB)
GWA-T-1 DAS-2	81	[1-128]	[1-4,295]	[10-51,070]	50	100	100
GWA-T-3 NorduGrid	34	1	[1-2,147]	[10-1,053,072]	24	1,400	50,000
GWA-T-4 AuverGrid	33	1	[1.7-3,668]	[10-259,316]	7	8,800	640,000
GWA-T-10 SHARCNET	147	[1-3000]	[1-32,021]	[10-2,087,029]	85	2,000	1,000
METACENTRUM-2009-2	42	[1-60]	[1-61,538]	[10-2,592,130]	44	100	20,000
PIK-IPLEX-2009-1	36	[1-2560]	[1-29,360]	[10-4,815,007]	88	89,000	4,700

online mechanisms, OVMPAC-X, and offline optimal mechanism, VCG-VMPAC. For the VCG-VMPAC mechanism, we use the CPLEX 12 solver provided by IBM to solve the VMPAC problem optimally. The mechanisms are implemented in C++ and the experiments are conducted on AMD 2.4GHz Dual Proc Dual Core nodes with 16GB RAM which are part of the WSU Grid System. In this section, we describe the experimental setup and analyze the experimental results.

A. Experimental Setup

Since real users request data have not been publicly released by cloud providers yet, we rely on well studied and standardized workloads from both, the Grid Workloads Archive [12], and the Parallel Workloads Archive [13]. The logs are selected based on the availability of recorded CPU and memory requests/usage. In our experiments, each job in a log represents a user request. We present statistics of the logs in Table II.

We consider each log as a series of requests, where the users can submit their requests over time to a cloud provider. We select 100 hours of the logs containing 706, 842, 1523, 1865, 677, and 416 requests for the selected logs, respectively. For each job in a log, we generate a user request. Since the logs provide data on resource usage, we consider these as values for the requested σ_{ir}, the amount of each resource of type r requested by user i, where i is a job in a log and r is a resource type. As a result, a user request contains the requested number of CPUs, the amount of memory and the amount of storage. To generate bids for users, we generate a random number b_i for each user i between 1 and 10. For OVMPAC-II, we use the job's runtime as the requested length of the job, while for OVMPAC-I, we set the requested length to one. As a result, the optimal offline results are different in each settings. We also generate a deadline for each job request which is between 3 to 6 times the job's runtime.

B. Analysis of Results

We compare the performance of OVMPAC-X and VCG-VMPAC for different workloads. For each workload, we record the execution time and the social welfare for each mechanism. We now present the results obtained by OVMPAC-I and OVMPAC-II for the selected logs.

1) OVMPAC-I: We analyze the performance of OVMPAC-I and VCG-VMPAC in terms of social welfare and execution time. Fig. 1a shows the social welfare for the selected logs. The results show that OVMPAC-I obtains a social welfare very close to that obtained by the optimal VCG-VMPAC mechanism. Fig. 1b shows the execution times of the mechanisms on a logarithmic scale. As we expected from the time complexity of the mechanism, the execution time of OVMPAC-I is very small. However, the execution time of the optimal offline mechanism, VCG-VMPAC, is more than six order of magnitudes greater than that of OVMPAC-I for each of the logs.

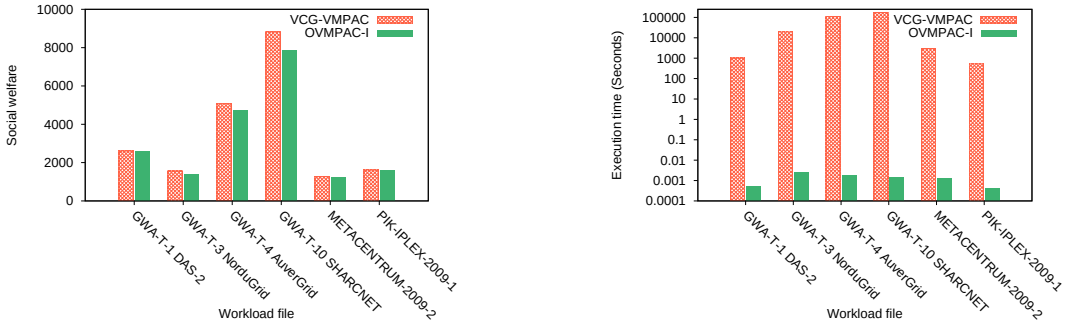
2) OVMPAC-II: We analyze the performance of OVMPAC-II and VCG-VMPAC in terms of social welfare and execution time. The optimal mechanism, VCG-VMPAC, could not find the solutions even after 72 hours for three out of the six logs. This is due to the fact that the problem becomes more complex for different job lengths, higher number of requests, and greater available capacity. Fig. 2a shows the social welfare achieved by the mechanisms. The results show that OVMPAC-II obtains a social welfare very close to that obtained by the optimal VCG-VMPAC mechanism. Fig. 2b shows the execution times of the mechanisms on a logarithmic scale. As we expected from the time complexity of the mechanisms, the execution time of OVMPAC-II is very small. However, the execution time of the optimal offline mechanism, VCG-VMPAC, is more than six order of magnitudes greater than that of OVMPAC-II for each of the logs.

In addition, the execution time of VCG-VMPAC for METACENTRUM-2009-2 in this setting ($l_i \geq 1$, $\forall i \in \mathcal{U}$) compared to that of the setting with $l_i = 1$ presented in Fig. 1b is one order of magnitude greater. This due to the characteristics of the requests of this log which makes the problem more complex for $l_i \geq 1$.

From the results of these experiments we can conclude that OVMPAC-X achieves a social welfare closer to the optimal (obtained by VCG-VMPAC). In addition to this OVMPAC-X decides the allocation much faster than VCG-VMPAC, thus making it very suitable for making allocation decisions in real-time.

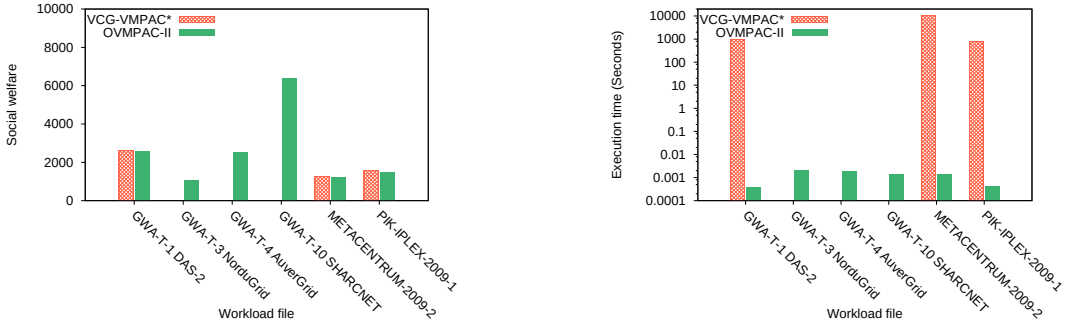
VI. Conclusion

We proposed online incentive-compatible mechanisms for VM provisioning and allocation in clouds that provide



(a) (b)

Figure 1: OVMPAC-I vs. VCG-VMPAC performance ($l_i = 1$): (a) Social welfare; (b) Execution time.



(a) (b)

Figure 2: OVMPAC-II vs. VCG-VMPAC Performance ($l_i \geq 1$): (a) Social welfare; (b) Execution time. (*VCG-VMPAC was not able to determine the allocation for GWA-T-3 NorduGrid, GWA-T-4 AuverGrid, and GWA-T-10 SHARCNET in feasible time, and thus, there are no bars in the plots for those cases)

incentives to the users to reveal their true valuations for the requested bundles of VM instances. We investigated the properties of our proposed mechanisms. The experimental results showed that the proposed online mechanisms obtain close to optimal social welfare and decide the allocation much faster than the offline optimal mechanism VCG-VMPAC, thus making them very suitable for deployment by cloud providers. For future work, we plan to design and investigate new monotone allocation functions that may lead to better performance for the online mechanisms.

Acknowledgment. This research was supported in part by NSF grants DGE-0654014 and CNS-1116787.

REFERENCES

[1] D. C. Parkes, "Online mechanisms," in *Algorithmic Game Theory*, N. Nisan, T. Roughgarden, Éva Tardos, and V. V. Vazirani, Eds. Cambridge University Press, 2007.

[2] A. Gershkov and B. Moldovanu, "Efficient sequential assignment with incomplete information," *Games and Economic Behavior*, vol. 68, no. 1, pp. 144–154, 2010.

[3] D. C. Parkes and S. Singh, "An MDP-based approach to online mechanism design," in *Proc. 17th Annual Conf. on Neural Information Processing Systems*, 2003.

[4] J.-J. Kuo, H.-H. Yang, and M.-J. Tsai, "Optimal approximation algorithm of virtual machine placement for data latency minimization in cloud systems," in *Proc. of IEEE INFOCOM*, 2014.

[5] L. M. Leslie, Y. C. Lee, P. Lu, and A. Y. Zomaya, "Exploiting performance and cost diversity in the cloud," in *Proc. of the 6th IEEE Intl. Conf. on Cloud Computing*, 2013, pp. 107–114.

[6] L. Mashayekhy, M. M. Nejad, and D. Grosu, "A truthful approximation mechanism for autonomic virtual machine provisioning and allocation in clouds," in *Proc. of the ACM Cloud and Autonomic Computing Conf.*, 2013, pp. 1–10.

[7] M. M. Nejad, L. Mashayekhy, and D. Grosu, "A family of truthful greedy mechanisms for dynamic virtual machine provisioning and allocation in clouds," in *Proc. of the 6th IEEE Intl. Conf. on Cloud Computing*, 2013, pp. 188–195.

[8] H. Zhang, B. Li, H. Jiang, F. Liu, A. V. Vasilakos, and J. Liu, "A framework for truthful online auctions in cloud computing with heterogeneous user demands," in *Proc. of IEEE INFOCOM*, 2013.

[9] S. Zaman and D. Grosu, "An online mechanism for dynamic vm provisioning and allocation in clouds," in *Proc. of the 5th IEEE Intl. Conf. on Cloud Computing*, 2012, pp. 253–260.

[10] N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani, *Algorithmic game theory*. Cambridge University Press, 2007.

[11] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*. Springer, 2004.

[12] Grid workloads archive. [Online]. Available: http://gwa.ewi.tudelft.nl

[13] Parallel workloads archive. [Online]. Available: http://www.cs.h-uji.ac.il/labs/parallel/workload/